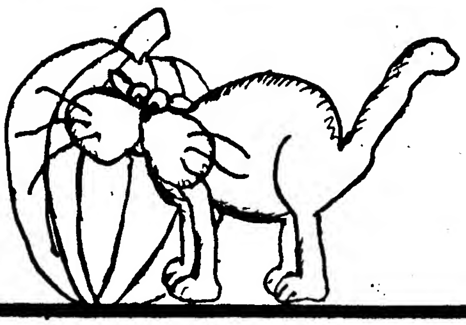
MOSAIC™ 64K SELECT NEWSLETTER

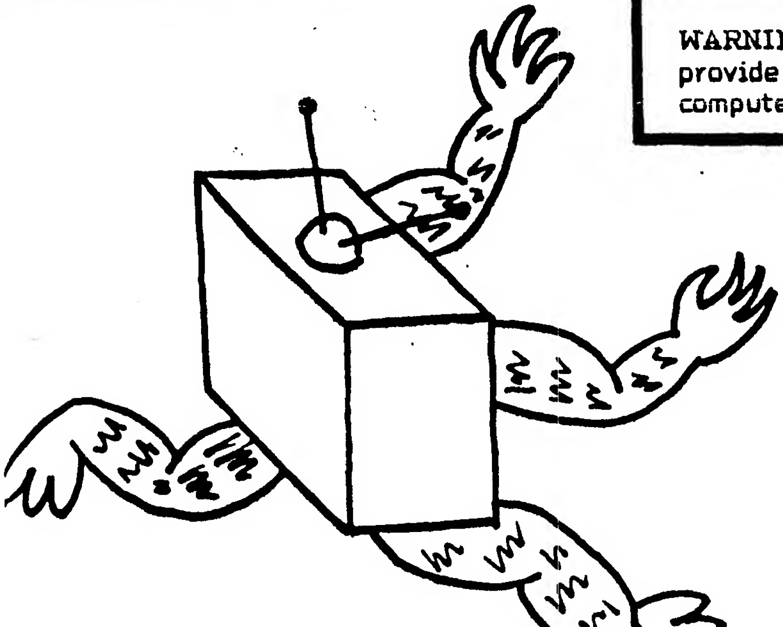


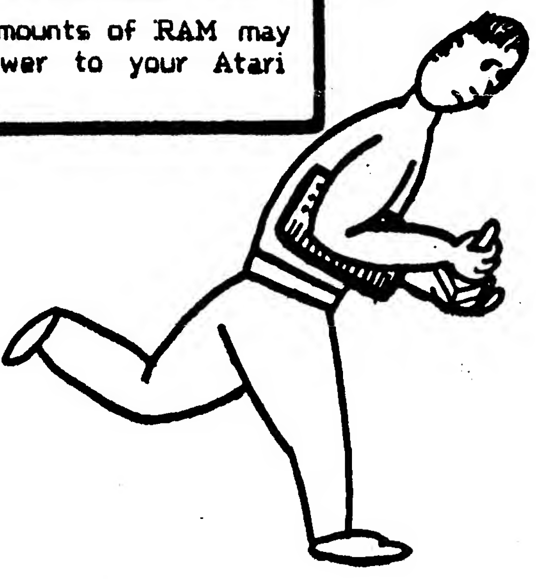
ISSUE #2

HAPPY HALLOWEEN HACKERS

From the Editors Desk

WARNING: Greater amounts of RAM may provide excessive power to your Atari computer!





Using Atari Microsoft Basic with the 64k RAM Select

By Gordon L. Banks

When the Atari Microsoft BASIC disk is booted, the Atari with the 64K Ram Select card installed will show a free memory of 23390. A 48K system will show a free memory of 21022. The difference is because of the additional 4K RAM available from the 64K board, but the difference isn't 4K. Why? What has happened? Where is the missing 1.7K of RAM?

When the Atari Microsoft Basic disk boots in, MEMTOP is checked and when it finds that the system is not 48K or less, something becomes confused and the BASIC code does not load in just where it should. The result is 1.7K less RAM than would normally be expected when using the 64K RAM Select board. If you want that last extra bit of RAM. here's how to get it.

First, boot up the system normally wth your Atari Microsoft BASIC disk. The free memory number should show as 23390 (22.8K). Now type POKE 4226,121 in the immediate mode. (Immediate mode is without typing a line number in front of the command. ED.) I'll explain what's going on a little later. Now type DOS to go to the DOS menu. Next, type L and RETURN. Answer the LOAD FROM... prompt with AUTORUN.SYS and press RETURN. Wait for the usual grinds and groans as the copy-protected program loads. When completed, you should be back in BASIC, but the free memory should now show 25118 (24.5K), 4K more than a 48K system.

One of the several copy-protection schemes employed in the Atari Microsoft BASIC disk is a second disk directory beginning with sector 377. Both directories contain the same filenames, but the AUTORUN.SYS file will not function from the directory located at sector 361. The DOS.SYS file contains the data on where the disk directory is located. Once DOS is loaded into RAM, the 'lo-byte' of the first disk directory sector number is located at 4226, with the 'hi-byte' located at 4229. Note that this is another exception to the normal 'lo-byte' and 'hi-byte' being located together. Earlier when we poked 4226 with 121 we were telling DOS to look for the disk directory 16 sectors higher in number. The normal disk directory is located at sectors 361 through 368, and location 4226 is normally a 105. Add this to the 'hi-byte' of 1 found in location 4229 and you get (105+256*1)=361, the first sector of the normal directory. By changing location 4226 to 16 higher, we informed DOS that we wanted the disk directory stored at 377. We then used that DOS to go to the menu and loaded AUTORUN.SYS there, which contained our Microsoft code. We did not have to change the disk directory location back again, as the AUTORUN.SYS file does it for us.

You can now take full advantage of the Mosaic 64K RAM Select board with your Atari Microsoft BASIC.

Ram Variable Relocater

By Winn Smith

Very few programs run out of memory simply by having too many instructions. In addition to storage for your program, you must also allow memory for the variables and strings that you use. In some applications, such as those that use large numerical or string arrays, the program could be very small and you still may run out of memory. The following subroutine can dramatically reduce the memory needed for string and data variable storage by storing those arrays in the bank-switched memory area of the Mosaic 64k RAM Select board located above the BASIC cartridge. Also, by using the bank switching features of the board, you may move large amounts of data between strings instantly by simply switching banks.

BASIC, uses a look-up table to determine where in memory various arrays are held. By changing the memory pointers, you can move the data storage area out of the standard program area. If the memory used is in the bank switched ram, switching a bank, will alter the data in the arrays with the data in the new bank of ram that was switched in.

The following listing demonstrates the uses of bank switched ram for storing variables:

```
10 DIM A$(1),B(1),C(1,1)
```

First, all arrays that are to be relocated into bank memory must be set to minimum size before any other variables are encountered.

```
20 VARCOUNT=3
```

Varcount is set to the number of arrays defined in line 10. In this case 3.

```
30 VVTP=PEEK(134)+PEEK(135)*256
40 STARP=PEEK(140)+PEEK(141)*256
```

These two lines determine the beginning location of the variable table, and the data storage area for the dimensoned arrays.

```
50 OFFSET=49152-STARP
```

OFFSET is the starting address of the new data storage area in the bank switched memory.

The data statements, define which type of arrays are used. The data statements must match the data arrays dimensioned in line 10.

60 DATA 0,20	Data type 0 is for string arrays. The second number in the data statement is the size of the string array which may be from 1 to 255. In the example shown, the string array would be equiv-ilent to DIM A$(20).
61 DATA 1,10	Data type 1 is for single dimensioned numrical arrays. In the example shown, the array would be equivilent to DIM B(10).
62 DATA 2,10,20	Data type 2 is for double dimensioned numerical ar-rays. In the example shown, the array would be equivi-lent to DIM C(10,20).
	Lines 70-340 make up a modi-fication loop which actually changes the memory locations in which the arrays are stored.
70 FOR VARS=0 TO VARCOUNT-1	The loop will run for as many times as there are variable arrays to change.
80 READ VARTYP 90 ON VARTYP GOTO 140,180	Read the variable type and decide which routine to use to modify the tables for that type of variable
	This routine processes type 0 arrays. (String arrays)
100 READ DM2 110 DM1=0	Get the string array length.
120 OFFCNT=DM2	Set OFFCNT to the number of characters allowed in the string.
130 GOTO 200	Go to the routine to store them in the proper places.
	This routine processes type 1 arrays. (Single dimen-sioned numerical arrays)
140 READ DM1	Get the size of the array.

`150 DM2=0`	
`160 OFFCNT=DM1*6`	Set OFFCNT to 6 times the size of the array, since each number requires 6 bytes of memory to store the data pointers
`170 GOTO 200`	Go to the routine to store them in the proper places.
	This routine processes type 3 arrays. (Double dimensioned numerical arrays)
`180 READ DM1,DM2`	Get the size of the array. There are two numbers used here to define a two dimensional array.
`190 OFFCNT=(DM1+DM2)*6`	Set OFFCNT to 6 times the combined dimensions of the array, since each number requires 6 bytes of memory to store the data pointers.

`200 OFFSHI=INT(OFFSET/256)`
`210 OFFSLO=OFFSET-OFFSETHI*256`
`220 DM1HI=INT(DM1/256)`
`230 DM1LO=DM1-DM1HI*256`
`240 DM2HI=INT(DM2/256)`
`250 DM2LO=DM2-DM2HI`
	Calculate the bytes to poke into the variable tables to update the variable pointers.

`260 TABENT=VVTP+8*VARCNT`	Determine the location of the table.

`270 POKE TABENT+2,OFFSLO`
`280 POKE TABENT+3,OFFSHI`
`290 POKE TABENT+4,DM1LO`
`300 POKE TABENT+5,DM1HI`
`310 POKE TABENT+6,DM2LO`
`320 POKE TABENT+7,DM2HI`
`330 OFFSET=OFFSET+OFFCNT`
	Poke in the new values to update the variable tables to refelct the new memory locations to store the data.

`340 NEXT VARS`	Loop until all the variable pointers have been changed.
	Continue your program on from here.

This program was written for simplicity. It can be greatly shortened by putting multiple statements per line and shortening variable names.